cupage

Release 0.8.2

August 16, 2014

Contents

`cupage` checks web pages and displays changes from the last run that match a given criteria. Its original purpose was to check web pages for new software releases, but it is easily configurable and can be used for other purposes.

It is written in Python, and requires v2.6 or later. `cupage` is released under the GPL v3

Contents:

CHAPTER 1

Background

I had been looking for a better way to help me keep on top of software releases for the projects I'm interested in, be that either personally or for things we use at work.

Some projects have Atom feeds, some have mailing lists just for release updates, some post updates on sites like freshmeat and some have no useful update watching mechanism at all. Tracking all these resources is annoying and a simple unified solution would be much more workable.

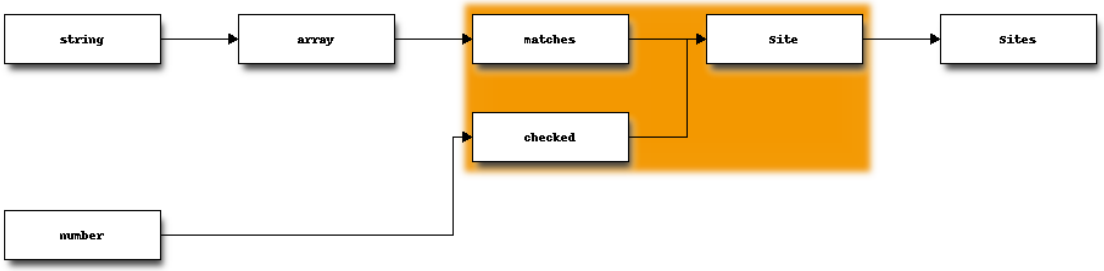
`cupage` is that solution, at least for my purposes. Maybe it could be for you too!

1.1 Database

With a local, unified tool we would instantly gain easy access to the updates database for use from other tools and applications.

JSON (JavaScript Object Notation) was chosen as it is simple to read and write, especially so from Python using the json module [1].

The database is a simple serialisation of the `cupage.Sites` object. The `cupage.Sites` object is a container for `cupage.Site` objects. Only persistent data from `cupage.Site` objects that can not be regenerated from the configuration file is stored in the database, namely last check time stamp and the current matches.



`matches` is an array, and contains the string matches of previous **cupage** runs.

`checked` is the offset in seconds from the Unix epoch that the site was last checked. It is normally a float, but may be `null`.

An example database file could be:

[1] Pickle was used in versions prior to 0.3.0. The switch was made as Pickle provided no benefits over JSON, and some significant drawbacks including the lack of support for reading it from other languages.

```
{
    "geany-plugins": {
        "matches": [
            "geany-plugins-0.17.1.tar.bz2",
            "geany-plugins-0.17.1.tar.gz",
            "geany-plugins-0.17.tar.bz2",
            "geany-plugins-0.17.tar.gz",
            "geany-plugins-0.18.tar.bz2",
            "geany-plugins-0.18.tar.gz"
        ],
        "checked": 1256677592.0
    },
    "interlude": {
        "matches": [
            "interlude-1.0.tar.gz"
        ],
        "checked": null
    }
}
```

CHAPTER 2

Configuration

cupage stores its configuration in `~/.cupage.conf` by default, although you can specify a different location with the *`cupage list -f`* command line option.

The configuration file is a `INI` format file, with a section for each site definition. The section header is the site's name which will be displayed in the update output, or used to select individual sites to check on the command line. Each section consists of a section of `name=value` option pairs.

An example configuration file is below:

```
[pep8]
site = pypi
match_type = tar
[pydelicious]
site = google code
match_type = zip
[pyisbn]
url = http://www.jnrowe.ukfsn.org/_downloads/
select = pre > a
match_type = tar
frequency = 6m
[upoints]
url = http://www.jnrowe.ukfsn.org/_downloads/
select = pre > a
match_type = tar
[fruity]
site = vim-script
script = 1871
[cupage]
site = github
user = JNRowe
frequency = 1m
```

Site definitions can either be specified entirely manually, or possibly with the built-in site matchers(see *site option* for available options).

2.1 `frequency` option

The `frequency` option allows you to set a minimum time between checks for specific sites within the configuration file.

The format is `<value> <units>` where value can be a integer or float, and units must be one of the entries from the table below:

Unit	Purpose
h	Hours
d	Days
w	Week
m	Month, which is defined as 28 days
y	Year, which is defined as 13 `m` units

2.2 `match` option

If `match_type` is `re` then `match` must be a valid regular expression that will be used to match within the selected elements. For most common uses a prebuilt `match_type` already exists(see *match_type option*), and `re` should really only be used as a last resort.

The Python re module is used, and any functionality allowed by the module is available in the `match` option(with the notable exception of the `verbose` syntax).

2.3 `match_type` option

The `match_type` value, if used, must be one of the following:

Match type	Purpose
`gem`	to match rubygems archives.
`re`	to define custom regular expressions
`tar`	to match gzip/bzip2/xz compressed tar archives(default)
`zip`	to match zip archives

The `match_type` values simply select a predefined regular expression to use. The base match is `<name>-[\d\.]+([_-](pre|rc)[\d]+)?\.<type>`, where `<name>` is the section name and `<type>` is the value of `match_type` for this section.

2.4 `select` option

The `select` option, if used, must be a valid CSS (Cascading Style Sheets) or XPath selector depending on the value of `selector` (see *selector option*) . Unless specified CSS CASCADING STYLE SHEETS) is the default selector type.

2.5 `selector` option

The `selector` option, if used, must be one of the following:

Selector	Purpose
css	To select elements within the page using CSS selectors (default)
xpath	To select elements within the page using XPath selectors

2.6 `site` option

The `site` option, if used, must be one of the following, hopefully self-explanatory values:

Site	Added	Required options
`cpan`	v0.4.0	
`debian`	v0.3.0	
`failpad`	v0.5.0	
`github`	v0.3.1	`user` (GitHub user name)
`google code`	v0.1.0	
`hackage`	v0.1.0	
`pypi`	v0.1.0	
`vim-script`	v0.3.0	`script` (script id on the vim website)

`site` options are simply shortcuts that are provided to reduce duplication in the configuration file. They define the values necessary to check for updates on the given site.

2.7 `url` option

The `url` value is the location of the page to be checked for updates. If used, it must be a valid FTP (File Transfer Protocol)/HTTP (HyperText Transfer Protocol)/HTTPS (HyperText Transfer Protocol) address.

CHAPTER 3

Usage

The **cupage** is run from the command prompt, and displays updates on `stdout`.

3.1 Options

--version
show program's version number and exit

--help
show this help message and exit

-v, --verbose
produce verbose output

-q, --quiet
output only matches and errors

3.2 Commands

3.2.1 `add` - add definition to config file

-f <file>, **--config** <file>
configuration file to read

-s <site>, **--site** <site>
site helper to use

-u <url>, **--url** <url>
site url to check

-t <type>, **--match-type** <type>
pre-defined regular expression to use

-m <regex>, **--match** <regex>
regular expression to use with –match-type=re

-q <frequency>, **--frequency** <frequency>
update check frequency

-x <selector>, **--select** <selector>
content selector

--selector <type>
selector method to use

3.2.2 `check` - check sites for updates

-f <file>, **--config** <file>
configuration file to read

-d <file>, **--database** <file>
database to store page data to. Default based on *--config* value, for example `--config my_conf` will result in a default setting of `--database my_conf.db`.

See *Database* for details of the database format.

-c <dir>, **--cache** <dir>
directory to store page cache

This can, and in fact *should* be, shared between all cupage uses.

--no-write
don't update cache or database

--force
ignore frequency checks

-t <n>, **--timeout**=<n>
timeout for network operations

3.2.3 `list` - list definitions from config file

-f <file>, **--config** <file>
configuration file to read

-m <regex>, **--match** <regex>
match sites using regular expression

3.2.4 `list-sites` - list supported site values

3.2.5 `remove` - remove site from config

-f <file>, **--config** <file>
configuration file to read

CHAPTER 4

cupage.py

4.1 Check for updates on web pages

Author James Rowe <jnrowe@gmail.com>

Date 2010-01-23

Copyright GPL v3

Manual section 1

Manual group Networking

4.2 SYNOPSIS

cupage.py [option]... <command>

4.3 DESCRIPTION

`cupage` checks web pages and displays changes from the last run that match a given criteria. Its original purpose was to check web pages for new software releases, but it is easily configurable and can be used for other purposes.

4.4 OPTIONS

--version	show program's version number and exit
--help	show this help message and exit
-v, --verbose	produce verbose output
-q, --quiet	output only matches and errors

4.5 COMMANDS

4.5.1 add

Add definition to config file

-f <file>, --config <file> configuration file to read

-s <site>, --site <site> site helper to use

-u <url>, --url <url> site url to check

-t <type>, --match-type <type> pre-defined regular expression to use

-m <regex>, --match <regex> regular expression to use with –match-type=re

-q <frequency>, --frequency <frequency> update check frequency

-x <selector>, --select <selector> content selector

--selector <type> selector method to use

4.5.2 check

Check sites for updates

-f <file>, --config <file> configuration file to read

-d <file>, --database <file> database to store page data to. Default based on *cupage check -f* value, for example `--config my_conf` will result in a default setting of `--database my_conf.db`.

See *Database* for details of the database format.

-c <dir>, --cache <dir> directory to store page cache

This can, and in fact *should* be, shared between all cupage uses.

--no-write don't update cache or database

--force ignore frequency checks

-t <n>, --timeout=<n> timeout for network operations

4.5.3 list

List definitions from config file

-f <file>, --config <file> configuration file to read

-m <regex>, --match <regex> match sites using regular expression

4.5.4 list-sites

List supported site values

4.5.5 remove

Remove site from config

-f <file>, --config <file> configuration file to read

4.6 CONFIGURATION FILE

The configuration file, by default **~/.cupage.conf**, is a simple **INI** format file, with sections defining sites to check. For example:

```
[spill]
url = http://www.rpcurnow.force9.co.uk/spill/index.html
select = p a
[rails]
site = vim-script
script = 1567
```

With the above configuration file the site named **spill** will be checked at **http://www.rpcurnow.force9.co.uk/spill/index.html**, and elements matching the CSS selector **p a** will be scanned for tarballs. The site named **rails** will be checked using the **vim-script** site matcher, which requires only a **script** value to check for updates in the scripts section of **http://www.vim.org**.

Various site matchers are available, see the output of `cupage.py --list-sites`.

4.7 BUGS

None known.

4.8 AUTHOR

Written by James Rowe

4.9 RESOURCES

Home page: http://github.com/JNRowe/cupage

4.10 COPYING

CHAPTER 5

Frequently Asked Questions

Ask them, and perhaps they'll become frequent enough to be added here ;)

CHAPTER 6

API documentation

Note: The documentation in this section is aimed at people wishing to contribute to *cupage*, and can be skipped if you are simply using the tool from the command line.

6.1 Site

Note: The documentation in this section is aimed at people wishing to contribute to *cupage*, and can be skipped if you are simply using the tool from the command line.

`cupage.`**`SITES`** = {}

Site specific configuration data

`cupage.`**`USER_AGENT`** = **'cupage/0.8.2 (https://github.com/JNRowe/cupage/)'**

User agent to use for HTTP requests

class `cupage.`**`Site`**(*name*, *url*, *match_func='default'*, *options=None*, *frequency=None*, *robots=True*, *checked=None*, *matches=None*)

Initialise a new `Site` object.

Parameters

- **name** (*str*) – Site name
- **url** (*str*) – URL for site
- **match_func** (*str*) – Function to use for retrieving matches
- **options** (*dict*) – Options for `match_func`
- **frequency** (*int*) – Site check frequency
- **robots** (*bool*) – Whether to respect a host's `robots.txt`
- **checked** (*datetime.datetime*) – Last checked date
- **matches** (*list*) – Previous matches

`check`(*cache=None*, *timeout=None*, *force=False*, *no_write=False*)

Check site for updates.

Parameters

- **cache** (*str*) – `httplib2.Http` cache location

- **timeout** (*int*) – Timeout value for `httplib2.Http`
- **force** (*bool*) – Ignore configured check frequency
- **no_write** (*bool*) – Do not write to cache, useful for testing

find_default_matches(*content*, *charset*)
Extract matches from content.

Parameters

- **content** (*str*) – Content to search
- **charset** (*str*) – Character set for content

find_github_matches(*content*, *charset*)
Extract matches from GitHub content.

Parameters

- **content** (*str*) – Content to search
- **charset** (*str*) – Character set for content

find_hackage_matches(*content*, *charset*)
Extract matches from hackage content.

Parameters

- **content** (*str*) – Content to search
- **charset** (*str*) – Character set for content

find_sourceforge_matches(*content*, *charset*)
Extract matches from sourceforge content.

Parameters

- **content** (*str*) – Content to search
- **charset** (*str*) – Character set for content

static **package_re**(*name*, *ext*, *verbose=False*)
Generate a compiled `re` for the package.

Parameters

- **name** (*str*) – File name to check for
- **ext** (*str*) – File extension to check
- **verbose** (*bool*) – Whether to enable `re.VERBOSE`

static **parse**(*name*, *options*, *data*)
Parse data generated by `Sites.loader`.

Parameters

- **name** (*str*) – Site name from config file
- **options** (*configobj.ConfigObj*) – Site options from config file
- **data** (*dict*) – Stored data from database file

state
Return `Site` state for database storage.

class `cupage.`**Sites**
`Site` bundle wrapper.

load(*config_file*, *database=None*)
Read sites from a user's config file and database.

Parameters

- **config_file** (*str*) – Config file to read
- **database** (*str*) – Database file to read

save(*database*)
Save `Sites` to the user's database.

Parameters **database** (*str*) – Database file to write

6.1.1 Examples

Reading stored configuration

```
>>> sites = Sites()
>>> sites.load('support/cupage.conf', 'support/cupage.db')
>>> sites[0].frequency
360000
```

Writing updates

```
>>> sites.save('support/cupage.db')
```

6.2 Command line

Note: The documentation in this section is aimed at people wishing to contribute to *cupage*, and can be skipped if you are simply using the tool from the command line.

cupage.cmdline.**USAGE = '%(prog)s checks web pages and displays changes from the last run that match\na given criteria.**
Command line help string, for use with `argparse`

cupage.cmdline.**main**()
Main script handler.

cupage.cmdline.**add**()
Commands are the basic building block of command line interfaces in Click. A basic command handles command line parsing and might dispatch more parsing to commands nested below it.

Changed in version 2.0: Added the *context_settings* parameter.

Parameters

- **name** – the name of the command to use unless a group overrides it.
- **context_settings** – an optional dictionary with defaults that are passed to the context object.
- **callback** – the callback to invoke. This is optional.
- **params** – the parameters to register with this command. This can be either `Option` or `Argument` objects.
- **help** – the help string to use for this command.

- **epilog** – like the help string but it's printed at the end of the help page after everything else.
- **short_help** – the short help to use for this command. This is shown on the command listing of the parent command.
- **add_help_option** – by default each command registers a `--help` option. This can be disabled by this parameter.

`cupage.cmdline.`**`check`**`()`

Commands are the basic building block of command line interfaces in Click. A basic command handles command line parsing and might dispatch more parsing to commands nested below it.

Changed in version 2.0: Added the *context_settings* parameter.

Parameters

- **name** – the name of the command to use unless a group overrides it.
- **context_settings** – an optional dictionary with defaults that are passed to the context object.
- **callback** – the callback to invoke. This is optional.
- **params** – the parameters to register with this command. This can be either `Option` or `Argument` objects.
- **help** – the help string to use for this command.
- **epilog** – like the help string but it's printed at the end of the help page after everything else.
- **short_help** – the short help to use for this command. This is shown on the command listing of the parent command.
- **add_help_option** – by default each command registers a `--help` option. This can be disabled by this parameter.

`cupage.cmdline.`**`list_conf`**`()`

Commands are the basic building block of command line interfaces in Click. A basic command handles command line parsing and might dispatch more parsing to commands nested below it.

Changed in version 2.0: Added the *context_settings* parameter.

Parameters

- **name** – the name of the command to use unless a group overrides it.
- **context_settings** – an optional dictionary with defaults that are passed to the context object.
- **callback** – the callback to invoke. This is optional.
- **params** – the parameters to register with this command. This can be either `Option` or `Argument` objects.
- **help** – the help string to use for this command.
- **epilog** – like the help string but it's printed at the end of the help page after everything else.
- **short_help** – the short help to use for this command. This is shown on the command listing of the parent command.
- **add_help_option** – by default each command registers a `--help` option. This can be disabled by this parameter.

`cupage.cmdline.`**`list_sites`**`()`

Commands are the basic building block of command line interfaces in Click. A basic command handles command line parsing and might dispatch more parsing to commands nested below it.

Changed in version 2.0: Added the *context_settings* parameter.

Parameters

- **name** – the name of the command to use unless a group overrides it.
- **context_settings** – an optional dictionary with defaults that are passed to the context object.
- **callback** – the callback to invoke. This is optional.
- **params** – the parameters to register with this command. This can be either `Option` or `Argument` objects.
- **help** – the help string to use for this command.
- **epilog** – like the help string but it's printed at the end of the help page after everything else.
- **short_help** – the short help to use for this command. This is shown on the command listing of the parent command.
- **add_help_option** – by default each command registers a `--help` option. This can be disabled by this parameter.

`cupage.cmdline.`**`remove`**`()`

Commands are the basic building block of command line interfaces in Click. A basic command handles command line parsing and might dispatch more parsing to commands nested below it.

Changed in version 2.0: Added the *context_settings* parameter.

Parameters

- **name** – the name of the command to use unless a group overrides it.
- **context_settings** – an optional dictionary with defaults that are passed to the context object.
- **callback** – the callback to invoke. This is optional.
- **params** – the parameters to register with this command. This can be either `Option` or `Argument` objects.
- **help** – the help string to use for this command.
- **epilog** – like the help string but it's printed at the end of the help page after everything else.
- **short_help** – the short help to use for this command. This is shown on the command listing of the parent command.
- **add_help_option** – by default each command registers a `--help` option. This can be disabled by this parameter.

6.2.1 Examples

Parse command line options

```
>>> options, args = process_command_line()
```

6.3 Utilities

Note: The documentation in this section is aimed at people wishing to contribute to *cupage*, and can be skipped if you are simply using the tool from the command line.

class cupage.utils.**CupageEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, long, float or None. If skipkeys is True, such items are simply skipped.

If *ensure_ascii* is true (the default), all non-ASCII characters in the output are escaped with uXXXX sequences, and the results are str instances consisting of ASCII characters only. If ensure_ascii is False, a result may be a unicode instance. This usually happens if the input contains unicode strings or the *encoding* parameter is used.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation. Since the default item separator is ', ', the output might include trailing whitespace when indent is specified. You can use separators=(',', ': ') to avoid this.

If specified, separators should be a (item_separator, key_separator) tuple. The default is (', ', ': '). To get the most compact JSON representation you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

If encoding is not None, then all input strings will be transformed into unicode using that encoding prior to JSON-encoding. The default is UTF-8.

default(*obj*)

Handle `datetime` objects when encoding as JSON.

This simply falls through to `default()` if `obj` has no `isoformat` method.

Parameters **obj** – Object to encode

cupage.utils.**json_to_datetime**(*obj*)

Parse `checked` datetimes from `cupage` databases.

See *json.JSONDecoder*

Parameters **obj** – Object to decode

cupage.utils.**parse_timedelta**(*delta*)

Parse human readable frequency.

Parameters **delta** (*str*) – Frequency to parse

cupage.utils.**sort_packages**(*packages*)

Order package list according to version number.

Parameters **packages** (*list*) – Packages to sort

cupage.utils.**robots_test**(*http*, *url*, *name*, *user_agent='*'*)

Check whether a given URL is blocked by `robots.txt`.

Parameters

- **http** – `httplib2.Http` object to use for requests
- **url** (*str*) – URL to check
- **name** – Site name being checked
- **user_agent** (*str*) – User agent to check in `robots.txt`

The following three functions are defined for purely cosmetic reasons, as they make the calling points easier to read.

`cupage.utils.`**`success`**(*text*)
Format a success message with colour, if possible.

Parameters **text** (*str*) – Text to format

`cupage.utils.`**`fail`**(*text*)
Format a failure message with colour, if possible.

Parameters **text** (*str*) – Text to format

`cupage.utils.`**`warn`**(*text*)
Format a warning message with colour, if possible.

Parameters **text** (*str*) – Text to format

6.3.1 Examples

Output formatting

```
>>> success('well done!')
u'\x1b[38;5;10mwell done!\x1b[m\x1b(B'
>>> fail('unlucky!')
u'\x1b[38;5;9munlucky!\x1b[m\x1b(B'
```

CHAPTER 7

Alternatives

Before diving in and spitting out this package I looked at the alternatives below. If I have missed something please drop me a mail.

It isn't meant to be unbiased, and you should try the packages out for yourself. I keep it here mostly as a reference for myself, and maybe to help out people who are already familiar with one of the entries below so they can see where I'm coming from.

7.1 ck4up

ck4up is a small tool written in Ruby which scans webpages for updates by storing the hash of checked pages. It provides pretty much the same functionality as `cupage`, but in a slightly different manner.

The major differences are a lack of HTTP cache support, a more manual configuration method and no built in support for various hosting sites.

7.2 urlwatch

urlwatch is a great little tool, which sends you emails containing the differences in web pages. To some extent `cupage` is mostly a narrow subset of the functionality provided by `urlwatch`, and the functionality could have been implemented on top with a bunch of hooks.

In my opinion the disadvantages are a lack of HTTP cache support, the configuration requires users to write Python and no built in support for various hosting sites. The massive advantage is how configurable and hackable the tool can be thanks to the "config is a python script" design.

CHAPTER 8

Release HOWTO

8.1 Test

In the general case tests can be run via `nose2`:

```
$ nose2 -vv tests
```

When preparing a release it is important to check that `cupage` works with all currently supported Python versions, and that the documentation is correct.

8.2 Prepare release

With the tests passing, perform the following steps

- Update the version data in `cupage/_version.py`
- Update `NEWS.rst`, if there are any user visible changes
- Commit the release notes and version changes
- Create a signed tag for the release
- Push the changes, including the new tag, to the GitHub repository

8.3 Update PyPI

Create and upload the new release tarballs to PyPI:

```
$ ./setup.py sdist --formats=bztar,gztar register upload --sign
```

Fetch the uploaded tarballs, and check for errors.

You should also perform test installations from PyPI, to check the experience `cupage` users will have.

CHAPTER 9

Appendix

class `httplib2.`**`Http`**
Instance of `Http` from httplib2

CHAPTER 10

Indices and tables

- *genindex*
- *modindex*
- *search*

C